PGPUB-DOCUMENT-NUMBER: 20020049802

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020049802 A1

TITLE: Process executing method and resource accessing method in computer system

PUBLICATION-DATE: April 25, 2002

INVENTOR-INFORMATION:

| NAME | CITY | STATE | COUNTRY |
|---|---|---|---|
| RULE-47 | | | |
| Nakahara, Masahiko | Yokohama-shi | | JP |
| Iwasaki, Masaaki | Tachikawa-shi | | JP |
| Takeuchi, Tadashi | Yokohama-shi | | JP |
| Nakano, Takahiro | Yokohama-shi | | JP |
| Serizawa, Kazuyoshi | Hadano-shi | | JP |
| Taguchi, Shihoko | Kawasaki-shi | | JP |

US-CL-CURRENT: **709/103**

ABSTRACT:

A process executing method capable of performing multiprocessing by using a shared resource without impairing periodical drivability of processes designed for executing continuous media processing. When a process requests the use of the shared resource, abortion of that process is first disabled by the process itself by using an abort disable function and then preemption of the same process is disabled by means of a preempt control module, whereupon the process enters a processing executed by using the shared resource. Upon completion of the processing for the shared resource, the process is immediately set to a preempt-enabled state by means of a preempt control module. After completion of all the processings, the abort-disabled state is finally cleared by using a disabled-abort clear function. Upon occurrence of forcive termination of a process in the abort-disabled state thereof, execution of this process is continued until the abort-disabled state is cleared, and the process is terminated forcibly after the abort-disabled state has been cleared.

---------- KWIC ---------

Current US Classification, US Primary Class/Subclass - CCPR (1):
**709/103**

Summary of Invention Paragraph - BSTX (6):
[0005] The problem that the process of high priority is inhibited from execution in succession because of the lock secured by the process of low priority is referred to as the priority inversion problem. Concerning this priority inversion problem, reference may be made to "**PRIORITY INHERITANCE** PROTOCOLS: AN APPROACH TO REAL-TIME SYNCHRONIZATION" in IEEE TRANSACTIONS ON COMPUTERS, Vol. 39, No. 9, September 1990, pp. 1175-1185. In the conventional operating system known heretofore, when the priority inversion takes place, the

dormant process of low priority acquired and locked the shared resource is executed with the topmost priority in order to unlock the shared resource so that the process of high priority can be executed as early as possible.

US 20020049802A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0049802 A1**

**Nakahara et al.** (43) **Pub. Date: Apr. 25, 2002**

(54) **PROCESS EXECUTING METHOD AND RESOURCE ACCESSING METHOD IN COMPUTER SYSTEM**

(76) Inventors: **Masahiko Nakahara**, Yokohama-shi (JP); **Masaaki Iwasaki**, Tachikawa-shi (JP); **Tadashi Takeuchi**, Yokohama-shi (JP); **Takahiro Nakano**, Yokohama-shi (JP); **Kazuyoshi Serizawa**, Hadano-shi (JP); **Shihoko Taguchi**, Kawasaki-shi (JP)

Correspondence Address:
**ANTONELLI TERRY STOUT AND KRAUS**
**SUITE 1800**
**1300 NORTH SEVENTEENTH STREET**
**ARLINGTON, VA 22209**

(21) Appl. No.: **09/838,767**

(22) Filed: **Apr. 20, 2001**

**Related U.S. Application Data**

(62) Division of application No. 08/917,477, filed on Aug. 26, 1996, now patented.

(30) **Foreign Application Priority Data**

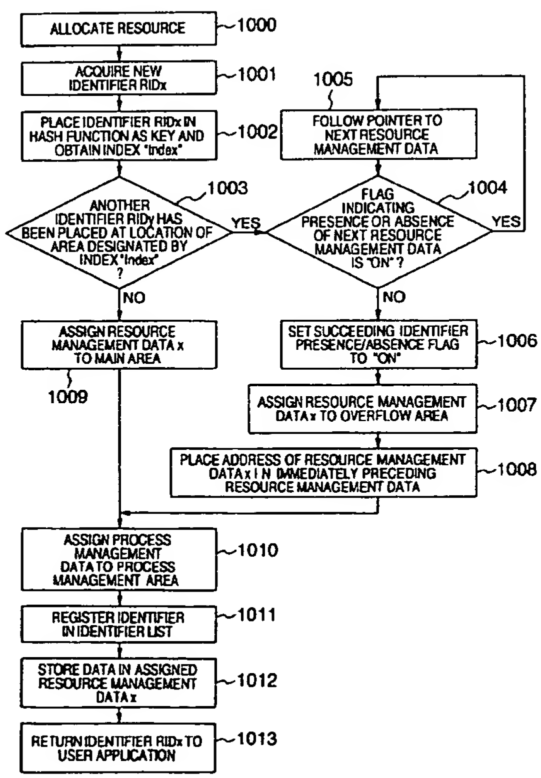
Aug. 28, 1996 (JP) .......................................... 08-226404
Dec. 6, 1996 (JP) .......................................... 08-326499

**Publication Classification**

(51) **Int. Cl.**[7] .................................................. **G06F 9/00**
(52) **U.S. Cl.** .................................................. **709/103**

(57) **ABSTRACT**

A process executing method capable of performing multiprocessing by using a shared resource without impairing periodical drivability of processes designed for executing continuous media processing. When a process requests the use of the shared resource, abortion of that process is first disabled by the process itself by using an abort disable function and then preemption of the same process is disabled by means of a preempt control module, whereupon the process enters a processing executed by using the shared resource. Upon completion of the processing for the shared resource, the process is immediately set to a preempt-enabled state by means of a preempt control module. After completion of all the processings, the abort-disabled state is finally cleared by using a disabled-abort clear function. Upon occurrence of forcive termination of a process in the abort-disabled state thereof, execution of this process is continued until the abort-disabled state is cleared, and the process is terminated forcibly after the abort-disabled state has been cleared.



US-PAT-NO: 6587955

DOCUMENT-IDENTIFIER: US 6587955 B1

TITLE: Real time synchronization in multi-threaded computer systems

DATE-ISSUED: July 1, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Foote; William | Cupertino | CA | N/A | N/A |
| Long; Dean Roy Ernest | Boulder Creek | CA | N/A | N/A |
| Fresko; Nedim | San Francisco | CA | N/A | N/A |

US-CL-CURRENT: 713/400, **709/107** , 714/39

ABSTRACT:

Methods and apparatus for implementing priority inversion avoidance protocols and deterministic locking where an API is used to select objects in a multi-threaded computer system are disclosed. In one aspect of the invention, an enhanced monitor is associated with one or more selected objects by way of an associated API. The enhanced monitor is arranged to set behavior for a lock associated with the selected objects as determined by a user defined behavior object included within the enhanced monitor. In this arrangement, only the selected one or more objects are associated with the enhanced monitor.

19 Claims, 7 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 6

---------- KWIC ---------

Brief Summary Text - BSTX (11):

There are several well-known priority inversion avoidance protocols which have been used extensively to prevent, or at least, substantially reduce the incidence and effect of priority inversions. One such approach to avoiding priority inversion, and more particularly, unbounded priority inversion is referred to as a **priority inheritance** protocol. As is well known in the art, **priority inheritance** means that when a thread waits on a mutex owned by a lower priority thread, the priority of the owner is increased to that of the waiter. In the **priority inheritance** protocol when a thread locks a mutex its priority is not changed. The action takes place when a thread attempts to lock a mutex owned by another thread. In this situation the priority of the thread owning the mutex is raised to the priority of the blocked thread (if higher). When the thread releases the mutex its old priority (i.e. prior to locking this mutex) is restored. This prevents unbounded priority inversion since the low priority thread gets a high priority and thus cannot be pre-empted by medium priority thread.

Brief Summary Text - BSTX (13):

Unfortunately, even though these priority inversion avoidance protocols have been successfully used to substantially reduce, and/or eliminate, the incidence of priority inversion, implementation in many computing systems it is prohibitively expensive in terms of both system and programmer resources in systems that contemplate very wide use of monitors. For example, in a Java based computing system each object has a lock associated with it. In this type of system, neither the priority ceiling nor the **priority inheritance** protocols can be efficiently implemented due to the large number of objects and associated locks included in the system. In order to implement either or both protocols in a Java based system, every object in the system would have to be associated with the protocol. This would represent a substantial investment in system resources since only a small fraction of all objects included within the system would actually benefit from implementation of priority inversion avoidance. As a result, any solution to priority inversion that imposes a significant overhead, in either time or space, on every object in a running system would be unacceptable.

Brief Summary Text - BSTX (19):
In some preferred embodiments, the priority inversion avoidance protocol is either a priority ceiling protocol or a **priority inheritance** protocol.

Detailed Description Text - DETX (3):
In one embodiment of the invention, the enhanced monitor provides the object with special monitor locking protocols designed to prevent priority inversion as well as providing deterministic locking. Such priority inversion avoidance protocols include both **priority ceiling and priority inheritance** protocols.

Detailed Description Text - DETX (8):
FIG. 3b is a diagrammatic representation of the interface between a prioritized thread for which timely execution is critical, an object, and an enhanced monitor in an object based system. A prioritized thread 310 attempts to execute a synchronization operation on an object 312 having an object header 314. In the described embodiment, the object 312 is identified as requiring synchronization semantics, or behavior, as provided by an enhanced monitor 316 by an ENHANCED_MONITOR flag field 318 included in the object header 314. The ENHANCED_MONITOR flag field 318 is used to identify the object 312 as one of the one or more objects included in the system for which behavior provided by the enhanced monitor 316 is important. Such behavior includes a special monitor lock that provides, for example, priority inversion avoidance protocols such as **priority ceiling or priority inheritance**. In this way, any excess system overhead is minimized and the advantages of implementing a desired priority inversion avoidance protocol is realized. In addition to the ENHANCED_MONITOR flag field 318, the object header 314 includes an enhanced monitor pointer field 320 that is used to identify a particular enhanced monitor that provides the locking behavior determined to be important for proper real time thread execution.

Claims Text - CLTX (7):
7. A method as recited in claim 6, wherein the priority inversion avoidance protocol is selected from the group comprising**: priority inheritance protocol and priority** ceiling protocol.

Claims Text - CLTX (15):
15. A method as recited in claim 14, wherein the priority inversion avoidance protocol is selected from the group comprising**: priority inheritance protocol and priority** ceiling protocol.

Current US Cross Reference Classification - CCXR (1):

**709/107**

Other Reference Publication - OREF (3):
Lui S., et al., "**Priority Inheritance** Protocols: An Approach to Real-Time Synchronization" IEEE Transactions on Computers, US, IEEE, Inc. NY, vol. 39, No. 9, Sep. 1, 1990, pp. 1175-1185.

US-PAT-NO: 6560628

DOCUMENT-IDENTIFIER: US 6560628 B1

TITLE: Apparatus, method, and recording medium for scheduling execution using time slot data

DATE-ISSUED: May 6, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Murata; Seiji | Tokyo | N/A | N/A | JP |

US-CL-CURRENT: 709/103, 709/100 , 709/102

ABSTRACT:


A scheduling method for use with a multi-thread system which is capable of time-sharing processing a plurality of threads is provided which can avoid the drawback of priority inversion, minimize the modification of a wait queue, and ensure the optimum use of the processing time of a CPU.

According to the present invention, a time slot data is assigned to each thread and the scheduling is carried out on the basis of the time slot data. As a processing time is imparted to the time slot data, the execution of the thread to which the time slot data is assigned is started. In case that a higher priority thread has to wait for the completion of the execution of a lower priority thread, the time slot data assigned to the higher priority thread is handled as a time slot data of the lower priority thread, hence allowing the execution of the lower priority thread to be started upon a processing time imparted to the time slot data.


12 Claims, 27 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 17

---------- KWIC ---------

Brief Summary Text - BSTX (11):

For eliminating the drawback of **priority inversion, priority inheritance schemes have been introduced (such as "Priority Inheritance** Protocols: An Approach to Real-time Synchronization" by Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky, Technical Report CMU-CS-87-181, the Computer Science Department of Carnegie Mellon University, November 1987). The **priority inheritance** scheme is designed in which when a higher priority thread is turned to its wait state due to the presence of a lower priority thread, the lower priority thread inherits the priority level of the higher priority thread.

Brief Summary Text - BSTX (12):

FIG. 2 illustrates a scenario, similar to that of FIG. 1, except for a **priority inheritance** scheme is employed. With the **priority inheritance** scheme, when the execution of a thread C is started with a thread A being in the wait state, the priority level of the thread C is changed to a level equal to the priority level of the thread A. As a result, interruption of the execution of

the thread C by the execution of a thread (for example, the thread B) of which the priority level is lower than that of the thread A will be prevented before the thread C leaves from a critical section CS1.

Brief Summary Text - BSTX (13):
The **priority inheritance** scheme allows the thread C to be scheduled at the priority level of the thread A before it leaves from the critical section CS1. Accordingly, the waiting time before the thread A is allowed to enter the critical section CS1 is bounded within a period required for the thread C occupying the critical section CS1. This permits an estimation of the interruption period during which the execution of the higher priority thread A is suspended.

Brief Summary Text - BSTX (14):
To implement the above described **priority inheritance** scheme, the following procedure has to be performed. It is noted that this procedure is explained for a case that the execution of the higher priority thread A is interrupted until the lower priority thread C explicitly leaves from the critical section CS1.

Brief Summary Text - BSTX (17):
By following the foregoing procedure, the **priority inheritance** scheme shown in FIG. 2 is implemented where the interruption of the execution of the thread C by a thread (for example, the thread B) of which the priority level is lower than that of the thread A can be avoided before the thread C leaves out from the critical section CS1.

Brief Summary Text - BSTX (18):
In a conventional scheduling system, the location in the wait queue of a thread to be queued is determined by its priority level at the time. Because the priority level is changed dynamically and frequently in the **priority inheritance** scheme, it is necessary to modify the wait queue at short intervals of time. Therefore, the use of the **priority inheritance** scheme requires modification or rescheduling of the wait queue to respond to the dynamic change of the priority level, hence resulting in increase of the overhead.

Brief Summary Text - BSTX (19):
More particularly, the **priority inheritance** scheme is effective to eliminate the drawback of priority inversion, but demands for dynamically changing the priority level at high frequency thus increasing the overhead due to rescheduling and declining the overall performance of the system.

Brief Summary Text - BSTX (22):
However, such rescheduling by calculating the processing time imparted to each thread results in increase of the overhead during the execution. Accordingly, the conventional **priority inheritance** scheme will hardly ensure the optimum use of a processing time of the CPU imparted to each thread.

Brief Summary Text - BSTX (26):
The scheduling apparatus of the present invention allows the time slot data to be designated separately of the executable subject and subjected to the scheduling, whereby when the executable subject of a higher priority level has to wait for the completion of the execution of the executable subject of a lower **priority level, the priority inheritance** can be carried out by implementing transfer of the time slot data which is low in the cost.

Brief Summary Text - BSTX (30):

The scheduling method of the present invention allows the time slot data to be designated separately of the executable subject and subjected to the scheduling, whereby when the executable subject of a higher priority level has to wait for the completion of the execution of the executable subject of a lower **priority level, the priority inheritance** can be carried out by implementing transfer of the time slot data which is low in the cost.

Brief Summary Text - BSTX (36):

Also, according to the present invention, the **priority inheritance** is implemented by handing the time slot data assigned to the higher priority thread as the time slot data of the lower priority thread. This resolves frequent modification of the wait queue in the **priority inheritance**. Therefore, the **priority inheritance** can be conducted without proceeding frequent modification of the wait queue which may increase undesirable overhead.

Drawing Description Text - DRTX (3):

FIG. 2 is a diagram explaining a **priority inheritance** scheme;

Drawing Description Text - DRTX (6):

FIG. 5 is a flowchart showing a procedure of calculating the function "changeOwner (Thread* target, Thread* depend)" called in "makeReady (Thread* target)"; while FIGS. 6 to 13 illustrating waiting patterns over a row of wait queues when no **priority inheritance** is involved,

Drawing Description Text - DRTX (16):

FIG. 15 is a diagram showing the time slot data A, B, and C shifted from the waiting pattern shown in FIG. 14; while FIGS. 16 to 25 illustrating waiting patterns over the wait queues when the **priority inheritance** is involved,

Detailed Description Text - DETX (5):

According to the present invention, the **inheritance of the priority** level is implemented by dynamically modifying the relation between a thread and its time slot data. More particularly, in case that a higher priority thread has to wait for the completion of the execution of a lower priority thread, the time slot data representing the higher priority thread is treated as the time slot data of the lower priority thread. When a processing time is then imparted to the latter time slot data, the execution of the lower priority thread is started.

Detailed Description Text - DETX (14):

The variable "inherit" is used when the priority level of the thread is transferred or inherited from another thread and set to a value indicating the another thread from which the priority level is inherited. In other words, the variable "inherit" expresses that the thread of interest is in its wait state to wait for the completion of the execution of the interrupting thread. If there is no **inheritance of the priority** level, the variable "inherit" is set to NULL.

Detailed Description Text - DETX (17):

In the thread data, "depend" and "inherit" represent link data to a piece of information related to the **priority inheritance**. For example, when the thread A is in the wait state before the execution of the thread B is completed, the variable "depend" in the thread data of the thread A and the variable "inherit" in the thread data of the thread B are associated with each other forming a two-directional link. It is noted that the variable in the thread data of a

thread is referred to as a variable of the thread in the following description. Also, the thread data may be called a thread for ease of the description.

Detailed Description Text - DETX (20):
In the scheduling described below, a **priority inheritance** is in effect conducted by changing a thread specified by a variable "owner" and hence by changing a thread for which the processing time of the CPU is imparted. More specifically, when a processing time of the CPU is imparted to the time slot data, the execution of the thread indicated by the variable "owner" of the time slot data is started. The **priority inheritance** is thus implemented by modifying the variable "owner" instead of directly operating on the wait queue.

Detailed Description Text - DETX (65):
4-1. When no **Priority Inheritance** is Involved

Detailed Description Text - DETX (66):
An example for time-sharing processing the thread A having a priority level of 15, the thread B having a priority level of 13, and the thread C having a priority level of 3 with no involvement of the **priority inheritance** is now explained referring to FIGS. 6 to 15. In FIGS. 6 to 15 and FIGS. 16 to 25, the numerals 1 to 17 represent the 17 wait queues respectively. The numeral with a circle mark indicates that the wait queue denoted by the numeral is to be examined.

Detailed Description Text - DETX (84):
4-2 When **Priority Inheritance** is Involved

Detailed Description Text - DETX (85):
An example when the **priority inheritance** is involved in the time-sharing processing of a thread A having a priority level of 15, a thread B having a priority level of 13, and a thread C having a priority level of 3 is now explained referring to FIGS. 16 to 26.

Detailed Description Text - DETX (104):
In the scheduling method of the present invention, when the higher priority thread has to wait for the completion of the execution of the lower priority thread, such troublesome operations as modifying the wait queue or recalculating the **priority level for priority inheritance are not needed to have the effect that the priority** level is transferred from the higher priority thread to the lower priority thread. Accordingly, the scheduling method is advanced to resolve the drawback of priority inversion which may be critical in the real-time system. Also, without modifying the wait queue nor recalculating the **priority level for priority inheritance,** the scheduling method will avoid increase of the cost in eliminating the drawback of priority inversion. Moreover, the scheduling method allows the processing time of the CPU imparted to a thread at its wait state to be assigned to an interrupting thread which causes the wait state of the thread, hence ensuring the optimum use of the processing time of the CPU without loss.

Detailed Description Text - DETX (105):
In the scheduling method of the present invention, when the wait state is repeated (for example, the thread A waiting for the execution of the thread B and the thread B waiting for the execution of the thread C), each corresponding time slot data is linked to the thread located at the top of the wait queue. Accordingly, the entire processing time of the CPU can be consumed at higher efficiency. More specifically, the waiting for a plurality of threads which is not allowed by the other conventional **priority inheritance** methods can

successfully be implemented.

Current US Original Classification - CCOR (1):
**709/103**

Current US Cross Reference Classification - CCXR (1):
**709/100**

Current US Cross Reference Classification - CCXR (2):
**709/102**

US-PAT-NO: 6560627

DOCUMENT-IDENTIFIER: US 6560627 B1

TITLE: Mutual exclusion at the record level with **priority inheritance** for embedded systems using one semaphore

DATE-ISSUED: May 6, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| McDonald; Michael F. | San Jose | CA | N/A | N/A |
| Arora; Sumeet | Milpitas | CA | N/A | N/A |
| Chu; Mark | Cupertino | CA | N/A | N/A |

US-CL-CURRENT: **709/103**, **709/100** , **709/104**

ABSTRACT:

A method for providing mutual exclusion at a single data element level for use in embedded systems. Entries for tasks that are currently holding a resource are stored in a hold list. Entries for tasks that are not currently executing and are waiting to be freed are stored in a wait list. A single mutual exclusion semaphore flags any request to access a resource.

24 Claims, 7 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 7

---------- KWIC ---------

TITLE - TI (1):
Mutual exclusion at the record level with **priority inheritance** for embedded systems using one semaphore

Brief Summary Text - BSTX (7):
Furthermore, in most applications, the mutual exclusion mechanism must support **priority inheritance**. If a low priority task holds a resource, and a higher priority task requests that resource, the priority of the low priority task should be elevated to that of the high priority task until it task releases the resource. Once the resource is released, priorities should revert to their original levels. In general, it is also desirable for the mutual exclusion mechanism to be able to detect and/or prevent deadlock. In a multi-tasking environment several tasks may compete for a finite number of resources. A task requests resources; if the resources are not available at that time the tasks enters the wait state. It may happen that waiting tasks will never again change state, because the resources they have requested are held by other waiting tasks. This situation is called deadlock. For example, deadlock occurs when a first task requests a record held by a second task while the second task is simultaneously requesting a record held by the first task. The result is neither task has its request answered. Such an occurrence could cause the application program or system software to crash.

Detailed Description Text - DETX (2):

A system is described that provides mutual exclusion of multiple tasks at the record level with **priority inheritance** and using one semaphore.

Detailed Description Text - DETX (8):

In one embodiment of the present invention, two lists are maintained within the system. One list is referred to as the "wait list" and contains a list of tasks that are not currently executing and are waiting for a resource to be freed. The second list is referred to as the "hold list" and contains a list of tasks that are currently holding a resource (e.g., a record). These queued lists are used to implement **priority inheritance** and deadlock prevention within the embedded system applications.

Detailed Description Text - DETX (20):

As described above, in one embodiment of the present invention, a **priority inheritance** scheme between two or more tasks is implemented. For example, in step 410 of FIG. 4, the current priority level of task B is shifted to that of task A. This **priority inheritance** mechanism is facilitated by the storage of each tasks' current and original priority in the hold list. The **inheritance of a higher priority** level by a lower priority task helps to ensure that the second task (the original lower priority task) executes quickly, and thus releases the resource quickly. **Priority inheritance** also prevents pre-emption of the second task by a third task. For example, by inheriting task A's priority level, task B will not get preempted by a third task ("task C"), where task C has a lower priority than task A but a higher level than task B's original level. Without such **priority inheritance** mechanism, task C could be allowed to run indefinitely without allowing task A to re-lock the resource temporarily held by task B.

Detailed Description Text - DETX (36):

In certain circumstances, it may occur that a referenced resource cannot be locked. FIG. 7 is a flowchart that illustrates the steps of locking multiple resources in which a referenced resource cannot be locked, according to one embodiment of the present invention. In the following discussion, it is assumed that a first resource ("resource R") cannot be locked. In step 702 the system unlocks all locked resources L in the reference list with lower priority than resource R. For each resource L, the system traverses the derived list and unlocks any resource derived from these resources, step 704. The system next unlocks all locked resources in the derived list with lower priority than resource R, step 706. In step 708 the current task is placed in the wait list. In step 710 **priority inheritance** among the competing tasks is checked. The system then releases the semaphore and suspends execution, step 712. When resource R becomes available, and the task waiting for it is resumed, the system then re-acquires the semaphore and continues execution, step 714. In most cases, the system will continue execution from step 602 in the process outlined in FIG. 6.

Detailed Description Text - DETX (38):

In the foregoing, a system has been described for providing mutual exclusion of multiple tasks at the record level with **priority inheritance** and using one semaphore. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention as set forth in the claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

Current US Original Classification - CCOR (1):
**709/103**

Current US Cross Reference Classification - CCXR (1):
**709/100**

Current US Cross Reference Classification - CCXR (2):
**709/104**

US-PAT-NO: 6301602

DOCUMENT-IDENTIFIER: US 6301602 B1
**See image for Certificate of Correction**

TITLE: Priority information display system

DATE-ISSUED: October 9, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Ueki; Katsuhiko | Tokyo | N/A | N/A | JP |

US-CL-CURRENT: 709/103, 709/100 , 709/102

ABSTRACT:

A priority information display system operates on an operating system that controls the execution of a plurality of processes having their respective **priorities and enables the inheritance of priority** between the plurality of processes. The system acquires from the operating system the priority information including information on the **inheritance of the priorities** of processes. Furthermore, on the basis of the acquired priority information, the system creates information on the coordinates of the priorities of processes on a coordinate system consisting of three axes of a time axis, a process type axis, and a priority value axis, transforms the coordinate information into coordinate information on a three-dimensional display coordinate system, and displays lines representing the **priorities of processes and the inheritance of priority** three-dimensionally.

18 Claims, 16 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 10

---------- KWIC ---------

Abstract Text - ABTX (1):

A priority information display system operates on an operating system that controls the execution of a plurality of processes having their respective **priorities and enables the inheritance of priority** between the plurality of processes. The system acquires from the operating system the priority information including information on the **inheritance of the priorities** of processes. Furthermore, on the basis of the acquired priority information, the system creates information on the coordinates of the priorities of processes on a coordinate system consisting of three axes of a time axis, a process type axis, and a priority value axis, transforms the coordinate information into coordinate information on a three-dimensional display coordinate system, and displays lines representing the **priorities of processes and the inheritance of priority** three-dimensionally.

Brief Summary Text - BSTX (4):

In recent years, an operating system with a **priority inheritance** algorithm that causes a process to inherit the priority of another process has been provided. In **priority inheritance by the algorithm, the present priority** of

the process that is to inherit priority is changed dynamically to the priority of the process from which the priority is inherited.

Brief Summary Text - BSTX (5):
Explanation will be given, provided that a single process has a "initial basic priority", a "basic priority", and a "present priority". The "initial basis priority" means the value of priority given to a process in the initial stage and remains unchanged, as a general rule. The "basic priority" means the value of priority that changes only due to a specific factor (a specific system call) excluding the **priority inheritance**, as a general rule. The "present priority" means the value of priority that the operating system uses in scheduling, as a general a rule. The value of the present priority changes, depending on various factors including **priority inheritance**.

Brief Summary Text - BSTX (6):
Some dynamic changes of **priority are related to inheritance and other dynamic changes of priority are not related to inheritance**. In the former case, that is, when dynamic changes of **priority are related to inheritance**, the factors triggering the change of priority include, for example, the issue of an interprocess synchronizing system call and the occurrence of an event, such as message exchange.

Brief Summary Text - BSTX (11):
In the environment for debugging concurrent programs executed on the operating system using the **priority inheritance** algorithm, the desired program is debugged by recording the history of events occurred during the execution of the program and observing the behavior of the process.

Brief Summary Text - BSTX (12):
The known methods of displaying information on priority, such as the current priority or the basic priority, include a method of recording the value of priority after the change at the time of issuing a priority change system call and a method of checking the priorities in unison with a specific timing. With these methods, however, the obtained priorities are only displayed in numerical values, which makes it impossible to display information on **priority inheritance** effectively.

Brief Summary Text - BSTX (17):
The foregoing objects are accomplished by providing a priority information display system which operates with an operating system that controls an execution of a plurality of processes having their respective **priorities and enables an inheritance of priority** between said processes, said priority information display system comprising:

Brief Summary Text - BSTX (18):
acquisition means for acquiring from said operating system a priority information including information on the **inheritance of the priorities** between said processes; and

Brief Summary Text - BSTX (20):
According to the present invention, when a specific event related to the priorities of processes has taken place, not only specific process information on the event can be acquired and displayed, but also the behavior of processes, the change of priority, the cause of priority change, etc. can be recorded and displayed to the user, so that the user can easily find bugs or the like in the application program stemming from **priority inheritance** caused by an unexpected

behavior of the process. Therefore, it is possible to provide a priority information display system that displays the program information effective in debugging the concurrent program.

Drawing Description Text - DRTX (7):
FIG. 5 is a flowchart to help explain the flow of processing at the time when **priority inheritance** has occurred;

Drawing Description Text - DRTX (8):
FIG. 6 shows an example of the pieces of **priority information (inheritance) recorded by the priority** information recording section 4;

Drawing Description Text - DRTX (9):
FIG. 7 is a flowchart to help explain the flow of the canceling process of **priority inheritance**;

Drawing Description Text - DRTX (10):
FIG. 8 shows an example of the pieces of **priority information (inheritance cancellation) recorded by the priority** information recording section 4;

Detailed Description Text - DETX (2):
Hereinafter, referring to the accompanying drawings, embodiments of a priority information display system according to the present invention will be explained. In a first embodiment, a system that acquires information on the priorities of processes, including **priority inheritance,** from the operating system and displays a list of pieces of the priority information will be explained. In a second embodiment, a system that displays the priority information three-dimensionally.

Detailed Description Text - DETX (6):
The operating system 1 has a function related to the above-described **priority inheritance** algorithm. Namely, the operating system 1 enables the **inheritance or cancellation of priority** between a plurality of processes. There may be a case where the function is provided as an additional function of the operating system 1.

Detailed Description Text - DETX (8):
The priority information acquisition section 3 holds the identifier for a process to be recorded that has been specified by the user and acquires from the operating system 1 information on the **priority of the process, priority inheritance,** and the change of the process state. According to the identifier for a process to be recorded, the section judges whether or not the acquired information is about the process to be recorded. If the information is about the process to be recorded, the section will send the acquired information to the priority information recording section 4.

Detailed Description Text - DETX (9):
Information (**priority inheritance** information) on a process related to **priority inheritance** includes the information used to let the user know that priority is inherited from a first process to a second process and that the priority given to the first process is canceled.

Detailed Description Text - DETX (13):
It is assumed that the application program (concurrent program) is made up

of, for example, three processes, process A, process B, and process C, which operate cooperatively using a **priority inheritance** semaphore (not shown). Here, explanation will be given as to how process B inherits the priority of process A and operates.

Detailed Description Text - DETX (22):
In the embodiment, the priority information acquisition section 3 is started by the notice from a **priority inheritance/priority** change/process state change sensing program previously built in the operating system 1. The sensing program is provided as event sensing means for sensing specific events concerning the priorities of processes, including **priority inheritance, priority** cancellation, basic priority change, and an event related to process state change.

Detailed Description Text - DETX (23):
Next, explanation will be given about the flow of processing in a case where **priority inheritance** has occurred in the course of executing the application program.

Detailed Description Text - DETX (24):
FIG. 5 is a flowchart to help explain the flow of processing at the time when **priority inheritance** has occurred.

Detailed Description Text - DETX (25):
When the **inheritance of priority** has taken place during the execution of the application program (ST51), the priority information acquisition section 3 judges whether or not the process that has inherited the priority is a process to be recorded (ST52). When having judged that the process that has inherited the priority is a process to be recorded, the priority information acquisition section 3 asks the operating system 1 about the **inheritance priority** and the identifier for the process (source process) from which the priority is inherited and acquires those pieces of information (ST53). Then, the priority information acquisition section 3 sends the **inheritance priority** information together with the process identifier and type of the process to be recorded to the priority information recording section 4 as **inheritance priority** information (ST54). The priority information recording section records the **inheritance priority** information (ST55).

Detailed Description Text - DETX (26):
FIG. 6 illustrates an example of the pieces of **inheritance priority** information recorded by the priority information recording section 4. The example of the figure shows that **priority inheritance** has occurred for process B (destination process) whose **inheritance priority** is 3 and the process that has inherited the priority is process A (source process).

Detailed Description Text - DETX (27):
Next, explanation will be given about the flow of processing in a case where the cancellation of **priority inheritance** has occurred during the execution of the application program.

Detailed Description Text - DETX (28):
FIG. 7 is a flowchart to help explain the flow of canceling the **priority inheritance**.

Detailed Description Text - DETX (29):

When the cancellation of the **priority inheritance** has taken place during the execution of the application program (ST71), the priority information acquisition section 3 judges whether or not the process for which the cancellation has occurred is a process to be recorded (ST72). When having judged that the process for which the cancellation has occurred is a process to be recorded, the priority information acquisition section 3 acquires the process identifier and type (inheritance cancellation) of the process and information on the cause of the cancellation and sends these pieces of information to the priority information recording section 4 (ST73). The priority information recording section 4 records them (ST74).

Detailed Description Text - DETX (30):
FIG. 8 shows an example of pieces of information on the cancellation of the **priority inheritance recorded by the priority** information recording section 4. The example of the figure shows that the cancellation of the **priority inheritance** has occurred for process B and the cause of the cancellation has come from semaphore A.

Detailed Description Text - DETX (39):
As described above, when the occurrence of **priority inheritance, the cancellation of priority inheritance, the change of the basic priority,** or the change of the process state has been occurred, the priority information recording section 4 manages these events by unique numbers. Then, the information necessary for displaying the priority information is read out at the request of the priority information display section 3.

Detailed Description Text - DETX (51):
As explained so far, with the embodiment, when specific events concerning the priorities and states of processes have taken place, pieces of specific priority information on the events can be acquired and displayed in list form. More specifically, in debugging a concurrent program, information on the priorities of processes to be debugged, including **inheritance, cancellation, and basic priority** change, can be recorded as a history and displayed in list form.

Detailed Description Text - DETX (52):
Therefore, with the embodiment, because the behavior of processes in the concurrent program, the change of priority, the cause of priority change, etc. can be recorded and displayed in list form for the user to understand it easily, the user can easily find bugs in the application program resulting from **priority inheritance** caused by an unexpected behavior of the process. This enables efficient debugging, improving the efficiency of the debugging work.

Detailed Description Text - DETX (72):
Reads in process information No. 13 that process A has caused process B (n=2) to inherit the priority. W-coordinate information No. 13 before **inheritance of the value of the priority** "(3, 2, -6), state Ready, before **priority inheritance**" and W-coordinate information No. 14 "(3, 2, -3), state Ready; process A **priority inheritance**" are obtained. From this time on, the **priority inheritance** state of process A and the before**-inheritance priority** 6 are stored as the priority state of process B.

Detailed Description Text - DETX (74):
Reads in process information No. 15 that semaphore A has caused process B (n=2) to cancel the **inheritance of the priority**. Because the fourth event has occurred, time t=4 is set and W-coordinate information No. 16 before the change of the priority value "(4, 2, -3), state Run, before cancellation of semaphore

A **priority inheritance**" and W-coordinate No. 17 "(4, 2, -6), state Run, cancellation of semaphore A **priority inheritance**" are obtained.

Detailed Description Text - DETX (87):
W-coordinate information No. 13: process B, (3, 2, -6), state Ready, before **priority inheritance**

Detailed Description Text - DETX (88):
W-coordinate information No. 14: process B, (3, 2, -3), state Ready, process A **priority inheritance**

Detailed Description Text - DETX (90):
W-coordinate information No. 16: process B, (4, 2, -3), state Run, before cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (91):
W-coordinate information No. 17: process B, (4, 2, -6), state Run, cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (108):
D-coordinate information No. 13: process B, d13, state Ready, before **priority inheritance**

Detailed Description Text - DETX (109):
D-coordinate information No. 14: process B, d14, state Ready, process A **priority inheritance**

Detailed Description Text - DETX (111):
D-coordinate information No. 16: process B, d16, state Run, before cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (112):
D-coordinate information No. 17: process B, d17, state Run, cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (126):
D-coordinate information No. 13: process B, d13, state Ready, before **priority inheritance**

Detailed Description Text - DETX (127):
D-coordinate information No. 14: process B, d14, state Ready, process A **priority inheritance**

Detailed Description Text - DETX (129):
D-coordinate information No. 16: process B, d16, state Run, before cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (130):
D-coordinate information No. 17: process B, d17, state Run, cancellation of semaphore A **priority inheritance**

Detailed Description Text - DETX (153):
According to information No. 14 and No. 12, a line representing **priority inheritance** is outputted in the range from d14 to d12.

Detailed Description Text - DETX (155):
As described above, the priority information display system of the second embodiment produces a three-dimensional image of priority information as shown in FIG. 15. In FIG. 15 (-3, -1, 7) is specified for the coordinate shift r by the user. The image enables the user to intuitively grasp the information on the process changing with time, including **priority inheritance,** cancellation, and process state change. In FIG. 15, a bold line represents the Run state of the process, a fine broken line represents the Ready state, and a denser fine broken line represents the Wait state. A broken-line arrow represents **priority inheritance** and a bold-line arrow represents process control. To make it easier to grasp the **priority inheritance** state, colored representation (hatching representation) is used. For instance, in the case of process B, it can be seen easily from the dense hatching that process B is in the **priority inheritance** state during the period of time during which the priority value remains at 3 (during the same period, the basic priority value remains at 6).

Detailed Description Text - DETX (156):
While in the above-described example, **priority inheritance** has taken place between process A and process B, a single process may inherit the priorities from a plurality of processes. FIG. 16 illustrates an example of display in such a case. As shown in the figure, the region indicated by R1 represents the state of **priority inheritance from process C (a priority** value of 5), and the region indicated by R2 represents the state of **priority inheritance from process B (a priority** value of 3). The R1 and R2 are hatched (colored) differently from each other, which enables the user to grasp the detailed information on **priority inheritance**.

Detailed Description Text - DETX (157):
As described until now, with the present invention, when a specific event related to the priorities of processes has taken place, not only specific process information on the event can be acquired and displayed in list form, but also the behavior of processes in a concurrent program, the change of priority, the cause of priority change, etc. can be recorded and displayed in list form to the user, so that the user can easily find bugs or the like in the application program stemming from **priority inheritance** caused by an unexpected behavior of the process. Therefore, it is possible to provide a priority information display system that displays the program information effective in debugging the concurrent program.

Claims Text - CLTX (1):
1. A priority information display system which operates with an operating system that controls an execution of a plurality of processes according to their respective **priorities and permits an inheritance of priority** between said processes, said priority information display system comprising:

Claims Text - CLTX (2):
a section configured to detect a **priority inheritance** event occurring in said operating system;

Claims Text - CLTX (3):
an acquisition section configured to acquire from said operating system, inheritance information relating to the **inheritance of priority which**

**corresponds to the priority inheritance** event; and

Claims Text - CLTX (4):
a display section configured to generate, based on the acquired inheritance information, display information which includes a three-dimensional line showing the **inheritance of priority** between the processes.

Claims Text - CLTX (7):
4. A priority information display system according to claim 1, wherein said acquisition section further acquires information on a cause of **priority cancellation according to said priority inheritance** event.

Claims Text - CLTX (8):
5. A priority information display system according to claim 1, further comprising a section configured to acquire information on the change of a basic **priority of a process as part of the inheritance** information.

Claims Text - CLTX (11):
8. A priority information display system which operates with an operating system that controls an execution of a plurality of processes according to their respective **priorities and permits an inheritance of priority** between said processes, said priority information display system comprising:

Claims Text - CLTX (12):
a section configured to detect a **priority inheritance** event occurring in said operating system;

Claims Text - CLTX (13):
an acquisition section configured to acquire, from said operating system, inheritance information relating to the **inheritance of priority which corresponds to the priority inheritance** event;

Claims Text - CLTX (16):
a display section configured to display three-dimensional lines representing the **priorities of the processes and the inheritance of priority** three-dimensionally according to the transformed coordinate information on the three-dimensional display.

Claims Text - CLTX (17):
9. A priority information display method with an operating system that controls an execution of a plurality of processes according to their respective **priorities and permits the inheritance of priority** between said processes, said priority information display method comprising:

Claims Text - CLTX (18):
detecting a **priority inheritance** event occurring in said operating system;

Claims Text - CLTX (19):
acquiring from said operating system inheritance information relating to the **inheritance of priority which corresponds to the priority inheritance** event; and

Claims Text - CLTX (20):
generating, based on the acquired inheritance information, display information which includes a three-dimensional line showing **priority inheritances** between the processes.

Claims Text - CLTX (23):
sensing an event related to the **inheritance of priority**; and

Claims Text - CLTX (24):
acquiring a **priority of inheritance** from another process and an identifier of the process from which the priority is inherited, according to said event sensing.

Claims Text - CLTX (26):
sensing an event related to the cancellation of **priority inheritance**; and

Claims Text - CLTX (32):
17. A priority information display method for use with an operating system that controls execution of a plurality of processes having respective **priorities and that enables the inheritance of priority** between processes, said priority information display method comprising:

Claims Text - CLTX (33):
acquiring, from said operating system, priority information including information on the **inheritance of process priorities**;

Claims Text - CLTX (36):
three-dimensionally displaying lines representing process **priorities and an inheritance of priority** according to the coordinate information on the three-dimensional display coordinate system.

Claims Text - CLTX (37):
18. A computer-readable medium having computer-executable instructions for performing, in association with an operating system that controls a plurality of processes having respective **priorities and permits priority inheritance** between processes, steps comprising:

Claims Text - CLTX (38):
detecting a **priority inheritance** event occurring in said operating system:

Claims Text - CLTX (39):
acquiring from said operating system inheritance information relating to the **inheritance of priority which corresponds to the priority inheritance** event;

Claims Text - CLTX (41):
generating, based on the acquired inheritance information, display information which includes a three-dimensional line representing the **inheritance of priority** between the processes.

Current US Original Classification - CCOR (1):
**709/103**

Current US Cross Reference Classification - CCXR (1):
**709/100**

Current US Cross Reference Classification - CCXR (2):
**709/102**

Other Reference Publication - OREF (1):
"**Priority Inheritance** Protocols: An Approach to Real-Time Synchronization", Lui Sha et al, IEEE 1990.

US-PAT-NO: 5944778

DOCUMENT-IDENTIFIER: US 5944778 A

TITLE: Periodic process scheduling method

DATE-ISSUED: August 31, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
| --- | --- | --- | --- | --- |
| Takeuchi; Tadashi | Yokohama | N/A | N/A | JP |
| Iwasaki; Masaaki | Tachikawa | N/A | N/A | JP |
| Nakahara; Masahiko | Yokohama | N/A | N/A | JP |
| Nakano; Takahiro | Kawasaki | N/A | N/A | JP |


US-CL-CURRENT: 709/100, 709/102

ABSTRACT:

A scheduling method of a periodic process of a computer system for keeping the execution interval of each process group constant as far as possible in the case where a plurality of process groups including periodically executed processes are executed in parallel. If a group master process of each process group specifies a wakeup interval period and a required CPU time per period and requests allocation of a CPU time, then a CPU allocation time of a specified process group is secured so as not to collide with a CPU allocation time of another process group, and a scheduling table is created so as to maintain the specified wakeup interval period. In response to arrival of time when one of the process groups should be waked up, a kernel process or a scheduler conducts wakeup by changing execution priority of a process belonging to this process group to highest priority ("raised") in the system, and thereafter maintains the highest priority for CPU allocation time consecutively allocated.

9 Claims, 22 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 18

---------- KWIC ---------

Detailed Description Text - DETX (261):

At step 2002, the priority of the periodic process 1202 functioning as the **inheritance source of the priority,** the priority of the periodic process 1202 functioning as the inheritance destination, and the counter field 1005 and the flag field 1006 in the process control block 1002 are updated. This updating method becomes similar to the flow chart shown in FIG. 11, and consequently it will not be described. Then, a jump to the step 2012 is effected.

Current US Original Classification - CCOR (1):

709/100

Current US Cross Reference Classification - CCXR (1):
<u>**709/102**</u>

US-PAT-NO: 5515538

DOCUMENT-IDENTIFIER: US 5515538 A
**See image for Certificate of Correction**

TITLE: Apparatus and method for interrupt handling in a multi-threaded operating system kernel

DATE-ISSUED: May 7, 1996

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Kleiman; Steven R. | Los Altos | CA | N/A | N/A |

US-CL-CURRENT: 710/260, **709/103** , **709/108**

ABSTRACT:

The disclosed invention is a method and apparatus for use in handling interrupts in a data processing system where the kernel is preemptible, has real-time scheduling ability, and which supports multithreading and tightly-coupled multiprocessors. The invention more specifically provides a technique for servicing interrupts in a processor by means of kernel interrupt handler threads which service the interrupt from start to finish. For efficiency, the interrupt handler threads do not require a complete context switch unless the interrupt handler thread is blocked. The kernel makes use of preprepared interrupt handler threads for additional efficiency, and these interrupt handler threads are not subjected to inordinate delays caused by the phenomenon of interrupt priority inversion if they do become blocked.

20 Claims, 12 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 12

---------- KWIC ---------

Detailed Description Text - DETX (10):

The mutex and writer locks support a dispatching **priority inheritance** protocol which prevents lower priority threads from blocking higher priority threads (priority inversions).

Detailed Description Text - DETX (61):

If the interrupt handler thread (thread B) does block (i.e. become blocked) because of some synchronizing condition, the processing which occurs is shown in FIG. 8. Referring to FIG. 8, the kernel puts the blocked thread B on the sleep queue of the synchronizing object related to the thread that is causing the block; and gives the dispatching priority level of the blocked interrupt handler thread B to the thread causing the block 182. This is called "**priority inheritance**" and insures that the blocking thread will run as soon as possible. Then the kernel transfers control to the dispatcher who sees that its register "t.sub.-- intr" shows that thread A is pinned. As a result, the dispatcher calls "thread.sub.-- unpin" 184. "Thread.sub.-- unpin" completes the context switch of thread A (the "pinned" thread); "Nulls" the "t.sub.-- intr" field of thread B; marks the interrupt level of the blocked interrupt thread B in the

"cpu.sub.-- intr.sub.-- active" field in the CPU's cpu structure, so that the interrupt priority level will be held by the interrupted CPU; and control is transferred to the dispatcher to initiate the highest priority thread. This is usually the blocking thread 186. At this point the blocked interrupt handler thread B awaits the end of the condition that caused it to block. When the synchronizing object controlling the sleep queue containing the sleeping thread B finally awakens thread B, the common interrupt code, which called the handler checks the "t.sub.-- intr" field and finds it equal to "Null" indicating thread B was blocked. 188 Thread B completes its handling of the interrupt and upon completion, and if no other blocks occur, returns (158 in FIG. 7b), and the common interrupt code puts the interrupt handler thread back onto the pre-prepared interrupt handler thread pool for that CPU and returns control to the dispatcher to switch to the highest priority runnable thread available. (176 in FIG. 7b).

Current US Cross Reference Classification - CCXR (1):
**709/103**

Current US Cross Reference Classification - CCXR (2):
**709/108**

US-PAT-NO: 5313633

DOCUMENT-IDENTIFIER: US 5313633 A
**See image for Certificate of Correction**

TITLE: Process and device for deciding class hierarchical relationship of object oriented language and process and device for determining method utilizing the same

DATE-ISSUED: May 17, 1994

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Tomita; Hiroshi | Kawasaki | N/A | N/A | JP |
| Yoshimura; Kiyozumi | Kawasaki | N/A | N/A | JP |

US-CL-CURRENT: 707/1, **709/315**


ABSTRACT:

A process and a device for deciding class hierarchical relationship of an object oriented language which, in decision of the relationship of class inheritance of an object oriented language having a hierarchical relationship of classes, performs coding processing for affixing the codes each having a predetermined relation to each of those classes and decides the relationship of inheritance in a hierarchy of two optional classes with said series of codes affixed to said two optional classes having said hierarchical relationship.


17 Claims, 28 Drawing figures

Exemplary Claim Number: 8

Number of Drawing Sheets: 26

---------- KWIC ---------

Detailed Description Text - DETX (48):

FIG. 22 shows an example of the class hierarchy and the method definition state of the present object oriented language In this example, a class f 131 has the class c 112, the class d 113 and the class e 114 as the direct super classes. The **priority of inheritance** is to be in the order of the relationship of inheritance 132 with the class c 112, the relationship of inheritance 133 with the class d 113 and the relationship of inheritance 134 with the class e 114.

Claims Text - CLTX (25):

17. The process of operating an object oriented language according to claim 16, wherein said converting into a single inheritance is made, for those classes that inherit the methods from a plurality of super classes, into a class hierarchy of single inheritance only from the super class having the highest **priority in the relationship of inheritance** of said plurality of super classes which is designated in advance.

Current US Cross Reference Classification - CCXR (1):

US-PAT-NO: 5313630

DOCUMENT-IDENTIFIER: US 5313630 A

TITLE: System of object oriented inheritance using the temporal status of superclasses

DATE-ISSUED: May 17, 1994

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Namioka; Miyoko | Sagamihara | N/A | N/A | JP |
| Satoh; Kazuhiro | Ebina | N/A | N/A | JP |
| Yamamoto; Youichi | Kawasaki | N/A | N/A | JP |
| Moki; Keiji | Kawasaki | N/A | N/A | JP |

US-CL-CURRENT: 707/103R, **709/315**

ABSTRACT:

An object-oriented data base management system connected to a plurality of data bases includes a class definition unit, a static inheritance processing unit included within the class definition unit, a message processing unit, a dynamic inheritance processing unit included within the message processing unit, and an object management unit. The object-oriented data base management system is of the type that defines the inheritance relationship between a plurality of classes and an arbitrary class and the **inheritance priority** order, the class being a basic unit of programming, and executes the data processing by solving the inheritance relationship between classes in accordance with the priority order. There is prepared, for each class, updatable inheritance solution status information which the static inheritance processing unit causes to be stored in a table. The inheritance solution status information includes status information and time information. The status information is representative of whether or not inheritance between the class and the superclasses can be completely solved, and the time information is representative of a time when the status is established. When a message processing is requested for an object of a class, the dynamic inheritance processing unit searches the method designated by the message while referring to the inheritance solution status information regarding the class and the superclass stored in the table.

7 Claims, 13 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 10

---------- KWIC ---------

Abstract Text - ABTX (1):

An object-oriented data base management system connected to a plurality of data bases includes a class definition unit, a static inheritance processing unit included within the class definition unit, a message processing unit, a dynamic inheritance processing unit included within the message processing

unit, and an object management unit. The object-oriented data base management system is of the type that defines the inheritance relationship between a plurality of classes and an arbitrary class and the **inheritance priority** order, the class being a basic unit of programming, and executes the data processing by solving the inheritance relationship between classes in accordance with the priority order. There is prepared, for each class, updatable inheritance solution status information which the static inheritance processing unit causes to be stored in a table. The inheritance solution status information includes status information and time information. The status information is representative of whether or not inheritance between the class and the superclasses can be completely solved, and the time information is representative of a time when the status is established. When a message processing is requested for an object of a class, the dynamic inheritance processing unit searches the method designated by the message while referring to the inheritance solution status information regarding the class and the superclass stored in the table.

Brief Summary Text - BSTX (20):
In order to achieve the above object, according to one aspect of the present invention, there is provided an object-oriented data processing and programming system which defines the inheritance relationship between a plurality of classes and an arbitrary class and the **inheritance priority** order of the plurality of classes, and executes the data processing by solving the **inheritance relationship of the class in accordance with the priority** order. The system is provided with a static inheritance processing unit, and a dynamic inheritance processing unit. The static inheritance processing unit causes a storage means to store inheritance solution status information for each class, the inheritance solution status information including status information representative of whether or not inheritance of all superclasses of an arbitrary class can be processed completely, and a status established time representative of the time when the status is established. The dynamic inheritance processing unit operates such that if a message processing of an arbitrary class is requested, it refers to the inheritance solution status information of the arbitrary class and the associated superclasses stored in the storage means, and searches a method designated by the message.

Brief Summary Text - BSTX (21):
According to the present invention, the static inheritance processing unit judges whether or not inheritance of all superclasses of each class can be processed completely, in accordance with the priority order of superclasses, and generates and updates the inheritance solution status information including the status information representative of the judgment results, and the status established time. For example, in updating the inheritance solution status information of a class, a superclass having a higher priority order is sequentially selected as a target superclass. The status established time of the class is compared with that of the target superclass. If the former time is earlier than the latter time, the inheritance processing of the target superclass is recurrently executed. After the inheritance processing of the target superclass, the definition contents of the target superclass are merged into the class. Alternatively, if the former time is later than the latter time, it means that the inheritance solution of the superclass has not been updated. Therefore, the recurrent inheritance processing of the target superclass and merge of the definition contents thereof are not carried out. After the above processes, the status of the target superclass is checked. If the status is the incomplete solution status, **inheritance of the superclass having a lower priority** order than that of the target superclass is omitted to terminate the inheritance processing of the class. If the status of the target superclass is the complete solution status, a superclass having the next highest priority order is selected to repeat the above processes.

Claims Text - CLTX (1):

1. A class inheritance solution process for an object-oriented data processing and programming system which defines inheritance relationships between a plurality of superclasses and each class which is a subclass of the superclasses, and the **inheritance priority** order of the superclasses, and executes the data processing by solving the **inheritance relationship of the class in accordance with the priority** order, said process comprising the steps of:

Claims Text - CLTX (23):
said class definition unit analyzes a class definition of a source form and generates a class object of a predetermined form, said class definition unit including a static inheritance processing unit which judges whether or not inheritance of a superclass designated in the class definition can be solved in accordance with the definitions regarding the inheritance relationship between the class and the superclass and the **inheritance priority order, and a dynamic inheritance** processing unit for searching for a method in accordance with a judgment given by said static inheritance processing unit;

Current US Cross Reference Classification - CCXR (1):
**709/315**